UNCLASSIFIED

SECURITY CLASSIFICATION OF THI

DTIC FILE COPY

(4)

REPORT DOC

READ INSTRUCTIONS BEFORE COMPLETING FORM

AD-A211 618

1. REPORT NUMBER
AI Memo 1134

:IPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Taxonomic Syntax for First Order Inference

'E OF REPORT & PERIOD COVERED
memorandum

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
David McAllester, Bob Given

8. CONTRACT OR GRANT NUMBER(s)
N00014-86-K-0180

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

12. REPORT DATE
June 1989

13. NUMBER OF PAGES
36

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Office of Naval Research
Information Systems
Arlington, VA 22217

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES
None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
taxonomy
theorem proving
knowledge representation
types
inference
automated reasoning.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Most knowledge representation languages are based on classes and taxonomic relationships between classes. Taxonomic hierarchies without defaults or exceptions are semantically equivalent to a collection of formulas in first order predicate calculus. Although designers of knowledge representation languages often express an intuitive feeling that there must be some advantage to representing facts as taxonomic relationships rather than first order for-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 cont.

mulas, there are few, if any, technical results supporting this intuition. We attempt to remedy this situation by presenting a taxonomic syntax for first order predicate calculus and a series of theorems that support the claim that taxonomic syntax is superior to classical syntax.

Taxonomic Syntax for First Order Inference

David McAllester
Robert Givan

A.I. Memo No. 1134 June, 1989

Abstract:
Most knowledge representation languages are based on classes and taxonomic relationships between classes. Taxonomic hierarchies without defaults or exceptions are semantically equivalent to a collection of formulas in first order predicate calculus. Although designers of knowledge representation languages often express an intuitive feeling that there must be some advantage to representing facts as taxonomic relationships rather than first order formulas, there are few, if any, technical results supporting this intuition. We attempt to remedy this situation by presenting a taxonomic syntax for first order predicate calculus and a series of theorems that support the claim that taxonomic syntax is superior to classical syntax.

Keywords: Taxonomy, Knowledge Representation, Theorem Proving, Types, Type Inference, Automated Reasoning.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part the National Science Foundation contract IRI-8819624 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-86-K-0180.

1 Introduction

Most knowledge representation languages are based on classes and taxonomic relationships between classes [Bobrow and Winograd, 1977], [Fahlman, 1979], [Brachman, 1983], [Brachman *et al.*, 1983]. Taxonomic hierarchies without defaults or exceptions are semantically equivalent to a collection of formulas in first order predicate calculus. Designers of knowledge representation languages have argued that there are computational advantages to representing facts as taxonomic relationships rather than first order formulas. However, these arguments are usually non-technical, appealing to the reader's intuition and common sense rather than technical analysis.

We define a taxonomic syntax for first order predicate calculus. In this syntax terms are generalized to the notion of a class expression. Each class expression denotes a subset of the first order domain and all atomic formulas are simple statements about class expressions. We show that the quantifier-free taxonomic literals, i.e. atomic formulas or their negations[1] are more expressive than literals of classical first order logic. For example, there exists a set of two quantifier-free taxonomic literals that is satisfiable but is not satisfied by any finite first order structure — any satisfiable set of literals in the classical predicate calculus with equality can be satisfied by some finite structure. In spite of the increased expressive power of taxonomic literals, we show that the satisfiability of any set of quantifier-free taxonomic literals is polynomial time decidable.

The two basic observations about taxonomic syntax—that quantifier-free taxonomic literals are more expressive than classical literals, and that the satisfiability of a set of quantifier-free taxonomic literals is polynomial time decidable—suggest that taxonomic syntax is more powerful, in some way, than classical syntax. However, these observations do not provide any clear way of taking advantage of taxonomic syntax in general theorem proving. To show the value of taxonomic syntax in general theorem proving, we define a "high-level" proof system based on a strengthened version of the decision procedure for the decidability of a set of quantifier-free taxonomic literals.

[1] In taxonomic syntax it is possible for atomic formulas to contain quantifiers; the decidability result only applies to sets of quantifier free taxonomic literals.

The strengthened decision procedure provides a technical notion of an "obvious" step in a mathematical proof; a high-level proof is a sequence of steps where each step obviously follows from previous steps.

There is a continuum between theorem verification and theorem proving. No modern theorem proving system can automatically find proofs of theorems as hard as the prime factorization theorem in number theory. A man-machine interactive system, however, can be used to verify such theorems [Bledsoe, 1977], [Boyer and Moore, 1979], [Constable *et al.*, 1985], [Ketonen, 1984] [McAllester, 1989]. Without powerful theorem proving mechanisms the amount of user-provided detail required is so large that non-trivial verifications are impractical. As the requirement for user-provided detail decreases, a verification system can make a continuous transformation from being a proof verifier to a proof finder. Thus the classification of systems into verifiers and provers is somewhat arbitrary. A high-level proof system combines the notion of a user-specified proof with the notion of a sophisticated theorem-proving procedure that determines the correctness of individual proof steps. The decision procedure for proof-step correctness should always terminate quickly.

Many of the features of the high-level proof system introduced here, such as focus objects and rules of obviousness, are independent of taxonomic syntax. These features of high-level proof systems were introduced by McAllester in the Ontic theorem verification system, [McAllester, 1989], and found to be effective in a machine verification of a proof of the Stone representation theorem for Boolean lattices from the axioms of Zermelo-Fraenkel set theory. The high-level proof system introduced by McAllester is not based on taxonomic syntax. In this paper we argue in favor of taxonomic syntax by comparing the length of high-level proofs in a system based on classical syntax with the length of proofs in an analogous system based on taxonomic syntax. We show that any proof in classical syntax can be translated into a proof of the same length in taxonomic syntax. Furthermore, we conjecture that the converse is not true, i.e., we conjecture that there exist proofs in taxonomic syntax such that all classical syntax proofs of the same result are much longer.

2 Taxonomic Syntax for First Order Logic

Our taxonomic syntax for first order logic is organized around classes and taxonomic formulas. Consider a model of first order logic. Each class expression of taxonomic syntax denotes a subset of the domain, or universe of discourse, of the first order model. The class expressions include ordinary first order terms as a special case. Under the semantics of taxonomic expressions, terms are class expressions that denote singleton sets. But there are many class expressions that are not terms in the ordinary sense. For example, a predicate symbol P of one argument is a class expression denoting the set of all objects in the first order domain that satisfy the predicate P. If $s_1 \ldots s_k$ are class expressions, and f is a function symbol which takes k arguments, then $f(s_1 \ldots s_k)$ is also a class expression and denotes the set of all elements which can be written as $f(x_1 \ldots x_k)$ where x_i is an element of the set denoted by s_i. Now consider a k-ary predicate symbol R, i.e., a predicate of k arguments. A predicate of k arguments can be viewed as a function which takes $k-1$ arguments and returns a set. More specifically, we can write $R(x_1 \ldots x_{k-1})$ to denote the set of all elements y such that $R(x_1 \ldots x_{k-1},\ y)$ is true. If $s_1 \ldots s_{k-1}$ are class expressions then $R(s_1 \ldots s_{k-1})$ is also a class expression and denotes the union of all sets of the form $R(x_1 \ldots x_{k-1})$ where x_i is an element of s_i. A class expression completely constructed from variables, constants, and function symbols will be called a term. Terms always denote singleton sets. In addition to the class expressions discussed above, taxonomic syntax allows for classes defined with formulas; one can construct a class expression that denotes the set of all objects x that satisfy an arbitrary formula $\Phi(x)$. In order to ensure that taxonomic syntax is expressively equivalent to classical first order logic, a distinguished class expression, *A-Thing*, always denoted the entire domain in any first order interpretation. For the sake of technical simplicity, we only allow interpretations with non-empty semantic domains. Thus the class *A-Thing* always denotes a non-empty set.

The formulas of taxonomic syntax include atomic statements about the taxonomic relationships between class expressions. More specifically, we write (**IS** s_1 s_2) to say that the class s_1 is a subset of the class s_2. We also write (**THERE-EXISTS** s) to say that the class s is non-empty and we write (**DETERMINED** s) to say that there is at most one element of the class s. Finally,

we write (INTERSECTS s t) to say that the class s has a non-empty intersection with the class t.

Definition: A *class expression* is either

- a variable,
- a constant symbol,
- a monadic predicate symbol,
- a k-ary function symbol applied to a k class expressions,
- a k-ary predicate symbol applied to $k-1$ class expressions,
- a such-that expression of the form (s x S.T. $\Phi(x)$) where s is a class expression, x is a variable, and $\Phi(x)$ is a taxonomic formula,
- or the distinguished class expression *A-Thing*.

A *taxonomic formula* is either

- an is-formula, (IS s_1 s_2), where s_1 and s_2 are class expressions,
- an existence-formula, (THERE-EXISTS s), where s is a class expression,
- a determined-formula, (DETERMINED s), where s is a class expression,
- an intersection-formula (INTERSECTS s t) where s and t are class expressions,
- or a Boolean combination of taxonomic formulas.

Formulas of the first four kinds will be called *atomic formulas*. A *literal* is either an atomic formula or the negation of an atomic formula. A formula or class expression is *quantifier-free* if it does not contain any such-that class expressions.

Given a model of first order logic and an interpretation of every variable as an element of the first order domain, each class expression in taxonomic

syntax can be unambiguously interpreted as a subset of the first order domain and each formula of taxonomic syntax can be assigned an unambiguous truth value. For example, the formula (IS x *A-Person*) is true just in case the value of the variable x is an element of the set denoted by the class expression *A-Person*. The formula (IS y *A-Child-of*(x)) is true just in case the pair $<x,\ y>$ is contained in the relation denoted by *A-Child-of*. The formula (IS z *A-Child-of*(*A-Child-of*(x))) is true just in case there exists some member y of the class *A-Child-of*(x) such that z is a member of the class *A-Child-of*(y). The formula (IS x *Times*(2 *A-Number*)) is true just in case x can be written as the product of 2 and some number, i.e., just in case x is an even number. The such-that class expression (*A-Person* x S.T. (THERE-EXISTS *A-Child-of*(x))) denotes the set of all people who have children.

Our definition of taxonomic formulas does not include classical quantification. All quantification is done with such-that class expressions. For example, the formula (THERE-EXISTS (*A-Person* x S.T. $\Phi(x)$)) is true just in case there exists some element x of the class *Person* such that $\Phi(x)$ is true. Universal quantification can be defined in terms of existential quantification and negation. Alternatively, one can express universal quantification directly with taxonomic atomic formulas. For example, (IS *A-Person* (*A-Person* x S.T. $\Phi(x)$)) is true if and only if $\Phi(x)$ is true for every member x of the set denoted by *A-Person*. The special class expression *A-Thing* ensures that one can quantify over the entire first order domain. For example, the classical formula $\exists x \Phi(x)$ is equivalent to the taxonomic formula (THERE-EXISTS (*A-Thing* x S.T. $\Phi'(x)$)) where $\Phi'(x)$ is the taxonomic translation of $\Phi(x)$.

3 Satisfiability of Quantifier-Free Taxonomic Literals

Every literal in classical first order logic with equality is semantically equivalent to some quantifier-free taxonomic literal. More specifically, note that classical terms are a subset of taxonomic class expressions -- any class expression constructed purely from constants and function symbols is syntactically a term of classical first order logic. For classical terms the IS relation is

semantically identical to equality, so any equation between classical terms is equivalent to a quantifier-free atomic formula of taxonomic syntax. However, most non-trivial quantifier-free taxonomic literals are not equivalent to any classical literal. For example, let P be a monadic predicate symbol and let f be a monadic function symbol. The pair of literals (IS P $f(P)$) and (NOT (IS $f(P)$ P)) is satisfiable. For example, P can be interpreted as the non-negative integers and f as the function that subtracts one from its argument. In this case $f(P)$ denotes the set containing the non-negative integers plus negative one. One can show, however, that this pair of literals cannot be satisfied by any *finite* first order structure. Every satisfiable set of literals in classical first order logic with equality can be satisfied by some finite structure.

Since quantifier-free taxonomic literals are more expressive than classical literals, it is not immediately clear whether or not one can efficiently determine the satisfiability of a set of quantifier-free taxonomic literals.

> **Taxonomic Quantifier-Free Decidability Theorem:** The satisfiability of a set of quantifier-free taxonomic literals is polynomial time decidable.

There is a well known corresponding theorem for classical first order logic; the satisfiability of a set of literals in first order logic with equality is polynomial time decidable. The classical decision procedure is based on the congruence closure algorithm [Kozen, 1977], [Downey *et al.*, 1980], [Nelson and Oppen, 1980]. Unfortunately, the taxonomic decision procedure is significantly more complex than the classical procedure based on congruence closure. To appreciate the complexity of the taxonomic satisfiability problem, consider the literals (IS $f(P)$ a), (IS $f(Q)$ b) and (NOT (IS a b)) where P and Q are monadic predicates, f is a monadic function and a and b are constant symbols. These literals imply that the classes P and Q must be disjoint: if c, say, was in both P and Q, then $f(c)$ must equal both a and b, contradicting the third literal. Now suppose we add the literals (IS c P), (IS $g(c)$ Q), (IS $g^6(P)$ P) and (IS $g^7(Q)$ Q) where c is a constant symbol, g is a monadic function symbol, and $g^n(s)$ abbreviates $g(g(\cdots g(s)))$ with n applications of g. All of these literals taken together are unsatisfiable. To see

this it suffices to observe that, under any interpretation, $q^{\mathcal{I}}(c)$ must be a member of both P and Q.

Any set of quantifier-free taxonomic literals can be efficiently translated into an equisatisfiable set of quantifier-free literals that does not contain existence, determined, or intersection-formulas. More specifically, both positive and negative literals involving existence, determined, and intersection-formulas can be replaced by literals involving is-formulas and new constant and function symbols. For example, the literal (NOT (INTERSECTS P Q)) can be translated into (IS $f(P)$ a), (IS $f(Q)$ b) and (NOT (IS a b)). Thus, without loss of generality, one can assume that every literal involves an is-formula. It turns out that this apparent simplification, i.e., the elimination of existence, determined, and intersection-formulas, is not a simplification at all. Our decision procedure relies on existence, determined, and intersection formulas. The decision procedure is based on the rules of inference listed in figure 1.

If Σ is a set of taxonomic literals the notation $\Sigma \vdash\!\!\circ\ \Psi$ abbreviates the statement that there exists a derivation of Ψ from Σ using the above rules of inference. The notation $\Sigma \vdash\!\!\circ\ \mathbf{F}$ abbreviates the statement that there exists some formula Ψ such that $\Sigma \vdash\!\!\circ\ \Psi$ and $\Sigma \vdash\!\!\circ$ (NOT Ψ). It is not clear that one can quickly determine whether or not $\Sigma \vdash\!\!\circ\ \Psi$, or whether $\Sigma \vdash\!\!\circ\ \mathbf{F}$. However, one can readily construct a decision procedure for a seemingly more restricted inference relation. More specifically, the notation $\Sigma \vdash \Psi$ abbreviates the statement that Ψ can be derived from Σ using the above rules *such that every class expression appearing in the derivation of* Ψ *also appears as a subexpression of some formula in* Σ. The notation $\Sigma \vdash \mathbf{F}$ abbreviates the statement that there exists a formula Ψ such that $\Sigma \vdash \Psi$ and $\Sigma \vdash$ (NOT Ψ). Section 4 gives a cubic procedure for determining if $\Sigma \vdash \mathbf{F}$. Section 5 contains a proof that if Σ is a set of quantifier-free taxonomic literals, and $\Sigma \not\vdash \mathbf{F}$, then Σ is satisfiable. This implies that $\Sigma \vdash \mathbf{F}$ if and only if $\Sigma \vdash\!\!\circ\ \mathbf{F}$ and thus the restricted relation is not really any weaker than the unrestricted relation.

(1) (THERE-EXISTS *A-Thing*)

(2) (IS s *A-Thing*)

(3) $\dfrac{\texttt{(IS } s_1\ t_1\texttt{)}, \cdots \texttt{(IS } s_n\ t_n\texttt{)}}{\texttt{(IS } R(s_1, \ldots s_n)\ R(t_1, \ldots t_n)\texttt{)}}$

(4) $\dfrac{\texttt{(IS } r\ s\texttt{)}, \texttt{(IS } s\ t\texttt{)}}{\texttt{(IS } r\ t\texttt{)}}$

(5) (IS t t)

(6) (THERE-EXISTS c)

(7) (DETERMINED c)

(8) $\dfrac{\texttt{(THERE-EXISTS } s_1\texttt{)} \ \ldots \ \texttt{(THERE-EXISTS } s_n\texttt{)}}{\texttt{(THERE-EXISTS } f(s1, \cdots s_n)\texttt{)}}$

(9) $\dfrac{\texttt{(DETERMINED } s_1\texttt{)}, \cdots \texttt{(DETERMINED } s_n\texttt{)}}{\texttt{(DETERMINED } f(s1, \cdots s_n)\texttt{)}}$

(10) $\dfrac{\texttt{(NOT (DETERMINED } t\texttt{))}}{\texttt{(THERE-EXISTS } t\texttt{)}}$

(11) $\dfrac{\texttt{(THERE-EXISTS } R(s_1, \ldots s_n)\texttt{)}}{\texttt{(THERE-EXISTS } s_i\texttt{)}}$

(12) $\dfrac{\texttt{(THERE-EXISTS } r\texttt{)}, \texttt{(IS } r\ t\texttt{)}}{\texttt{(THERE-EXISTS } t\texttt{)}}$

(13) $\dfrac{\texttt{(DETERMINED } t\texttt{)}, \texttt{(IS } r\ t\texttt{)}}{\texttt{(DETERMINED } r\texttt{)}}$

(14) $\dfrac{\texttt{(NOT (IS } r\ t\texttt{))}}{\texttt{(THERE-EXISTS } r\texttt{)}}$

(15) $\dfrac{\texttt{(THERE-EXISTS } r\texttt{)}, \texttt{(IS } r\ s\texttt{)}, \texttt{(IS } r\ t\texttt{)}}{\texttt{(INTERSECTS } s\ t\texttt{)}}$

(16) $\dfrac{\texttt{(INTERSECTS } r\ t\texttt{)}, \texttt{(IS } r\ s\texttt{)}}{\texttt{(INTERSECTS } s\ t\texttt{)}}$

(17) $\dfrac{\texttt{(INTERSECTS } r_1\ s_1\texttt{)}, \cdots \texttt{(INTERSECTS } r_n\ s_n\texttt{)}}{\texttt{(INTERSECTS } f(r_1, \ldots r_n)\ f(s_1, \ldots s_n)\texttt{)}}$

(18) $\dfrac{\texttt{(INTERSECTS } r\ s\texttt{)}}{\texttt{(INTERSECTS } s\ r\texttt{)}}$

(19) $\dfrac{\texttt{(INTERSECTS } r\ s\texttt{)}}{\texttt{(THERE-EXISTS } s\texttt{)}}$

(20) $\dfrac{\texttt{(INTERSECTS } s\ t\texttt{)}, \texttt{(DETERMINED } s\texttt{)}}{\texttt{(IS } s\ t\texttt{)}}$

Figure 1: The inference rules for quantifier-free literals. In these rules the letters s, r, and t range over class expressions, c ranges over constant symbols, f ranges over function symbols, and R ranges over both function and predicate symbols.

4 A Satisfiability Decision Procedure

Let Σ be a set of quantifier-free taxonomic literals and let Υ be the set of class expressions containing all class expressions that appear as subexpressions of members of Σ, plus the distinguished class expression *A-Thing*. The set Υ of class expressions can be viewed as a semantic network where the elements of Υ are viewed as nodes representing classes. The decision procedure for determining whether $\Sigma \vdash \mathbf{F}$ can be viewed as a label-propagation process on this network. More specifically, it is possible to show that if Ψ is a formula not in Σ, but $\Sigma \vdash \Psi$, then Ψ must be a label formula for Υ as defined below.

> **Definition:** A *label formula* for a set Υ of class expressions is a formula of the form (`THERE-EXISTS` s), (`DETERMINED` s), (`IS` s t), or (`INTERSECTS` s t) where s and t are members of Υ.

Since some of the label formulas involve two members of Υ, it is perhaps better to view them as arcs between nodes rather than labels on nodes. It is possible to determine whether or not $\Sigma \vdash \mathbf{F}$ by propagating labels on the network Υ. More specifically, one continues to derive new label formulas until no more such derivations can be made. If Υ contains n nodes then there are $O(n^2)$ label formulas. Thus the process of deriving new formulas must terminate. If this propagation process yields some label formula Ψ such that Σ contains (`NOT` Ψ), then $\Sigma \vdash \mathbf{F}$, otherwise $\Sigma \nvdash \mathbf{F}$.

To analyze the running time of the label propagation procedure it is necessary to specify the procedure in greater detail. In presenting the details of our decision procedure we assume that all class expressions that are applications of a relation or function symbol involve at most two arguments. Expressions involving more than two arguments can be reformulated in terms of expressions that involve only two arguments and thus there is no loss of generality in restricting applications to two arguments. More specifically, if there is a function f of more than two arguments then one simply introduces a new function symbol g and uniformly replaces every class expression of the form $f(s_1, s_2 \ldots s_n)$ with $f(s_1, g(s_2 \ldots s_n))$. If the new function g takes more than two arguments the process can be repeated. In the worst case this transformation process leads to a linear increase in the length of expressions.

Our procedure runs on a graph-like data structure where each node represents an expression in Υ. This graph-like data structure can be viewed as a directed acyclic graph (DAG) representation of the class expressions in Υ. Each node in this graph is a data structure containing various kinds of information. The data structure representing a class expression s contains fields that are updated whenever a formula of the form (THERE-EXISTS s) or (DETERMINED s) is derived. The data structure representing s also contains a list of all the nodes t such that the formula (IS s t) has been derived, as well as a list of all nodes w such that (IS w s) has been derived, and a list of all nodes u such that (INTERSECTS s u) has been derived. Each time a new label formula is added the procedure must check to see if this addition can be propagated to yield further additional label formulas. There is a propagation procedure for each kind of label formula. For example there is a propagation procedure that is called when a new formula of the form (IS s t) is derived and a different procedure that is called when a new formula of the form (THERE-EXISTS s) is derived.

Each inference rule is implemented by pieces of propagation procedures. Since there is no way of knowing which antecedent will be derived last, each antecedent of a given rule corresponds to a piece of one of the propagation procedures. For example, consider the third rule of the previous section, the monotonicity rule. For applications involving two arguments, the rule says that if one can derive (IS s t) and (IS u w), then one can derive (IS $R(s,u)$ $R(t,w)$). Each of the two antecedents of this rule corresponds to a piece of the procedure for propagating new is-formulas. Consider the first antecedent, (IS s t). When a new formula (IS s t) is derived a certain piece of the procedure for propagating is-formulas finds all expressions in Υ of the form $R(s,u)$. Expressions of the form $R(s,u)$ are stored on a list in the data structure representing s. For each previously derived formula of the form (IS u w), a hash table lookup is used to see if the expression $R(t,w)$ is in Υ. If so, the formula (IS $R(s,u)$ $R(t,w)$) is derived and, provided that this formula has not been previously derived, the is-formula propagation procedure is called recursively on the new formula. Since there is no way of knowing which antecedent of the rule will be derived last, there is also a piece of the procedure for propagating is-formulas that corresponds to the second antecedent. When a new is-formula (IS u w) is derived, this piece finds all expressions in Υ of the form $R(s,u)$ and then for each previously

derived formula (IS s t) looks for the expression $R(t, w)$ in a hash table. This may lead to the recursive addition of another is-formula. Each of the other rules can also be implemented with pieces of propagation procedures; one piece for each antecedent of the rule. Rule 12, for example, can be implemented as a piece of the procedure for propagating existence formulas and a piece of the procedure for propagating is-formulas. Rule 17 is analogous to monotonicity rule and is implemented by pieces of the procedure for propagating intersection-formulas. The propagation procedures are recursive and no queue of outstanding inferences is required.

The total running time of the propagation process is equal to the sum over all rules of the time spent executing the pieces of the propagation procedures that correspond to that rule. For example, consider the monotonicity rule as discussed above. Assuming that hash table lookups take constant time, the time spent executing the monotonicity pieces of the is-formula propagation procedure is bounded by some constant times the total number of hash table lookups performed by these pieces. It is possible to show that for each term $R(s, u)$ in Υ, and each pair of derived is-formulas of the form (IS s t) and (IS u w), there is exactly one hash table lookup performed by the monotonicity pieces of the is-formula propagation procedure; at the point where both is-formulas are derived the expression $R(t, w)$ will be looked up in the hash table. For a fixed expression $R(s, u)$ in Υ, the propagation process can derive at most n^2 pairs of is-formulas of the form (IS s t) and (IS u w). Therefore, there are at most n^3 hash table lookups performed in the monotonicity pieces of the is-formula propagation procedure.

Assuming that no application expression has more than two arguments, each rule can be implemented so that at most $O(n^3)$ time is spent in the pieces of the propagation procedures that correspond to that rule (where n is the number of class expressions in Υ). Thus, if applications involve at most two arguments, the total time spent in the propagation process is at most $O(n^3)$.

5 Correctness of the Decision Procedure

Suppose that Σ is a set of quantifier-free taxonomic literals. This section summarizes a proof that if $\Sigma \not\vdash \mathbf{F}$ then Σ is satisfiable and thus the procedure of the previous section can determine the satisfiability of Σ.[2] The proof is based on a method for constructing a model of Σ from the set of label formulas Ψ such that $\Sigma \vdash \Psi$. As pointed out earlier, it is possible that Σ is satisfiable and yet there are no finite models of Σ. Thus, the method of constructing a model of Σ must be capable of yielding infinite models. However, the structure of the model is somehow completely characterized by the finite set of label formulas Ψ such that $\Sigma \vdash \Psi$.

Let Υ be the set of class expressions containing all class expressions appearing as subexpressions of formulas in Σ plus the distinguished class symbol *A-Thing*. The domain elements in any interpretation of Σ can be classified into types depending on their relationships with the class expressions in Υ. More specifically, if d is a domain element of a model of Σ, then the Υ-type of d is defined to be the set of class expressions s in Υ such that d is contained in the set denoted by s. If we view the class expressions in Υ as predicates, then the Υ-type of d is the set of class expressions that are true of d. More generally, an Υ-type is defined to be any subset of the class expressions in Υ. If there are n class expressions in Υ, then there are 2^n different Υ-types. We say that an Υ-type τ is *inhabited* in a particular model of Σ if there exists some domain element d of that model whose Υ-type is τ. Of course, there can be models in which many of the Υ-types are not inhabited.

The model we construct will have the property that existence formulas and intersection formulas that are not derivable by label propagation will be false in the model. This condition places constraints on the Υ-types that can be inhabited in our model. The types consistent with these constraints are said to be Σ-inhabitable. More specifically, a Σ-inhabitable Υ-type is an

[2] We have found two different proofs of this result: one presented in this section and another proof based on a syntactic proof that $\vdash$ is the same as $\vdash_o$ plus a semantic proof that $\vdash_o$ is complete for the detection of unsatisfiability. The syntactic proof that $\vdash$ is the same as $\vdash_o$ is somewhat complex but similar to the proof given in section 7. The proof that $\vdash_o$ is semantically complete for detecting the unsatisfiability of quantifier-free taxonomic literals is considerably simpler than the direct semantic proof for $\vdash$ given here.

Υ-type τ such that

- τ contains the type *A-Thing*,
- $\Sigma \vdash$ (THERE-EXISTS s) for every s in τ,
- if s is in τ and $\Sigma \vdash$ (IS s w) then w is in τ,
- and for all s and w in τ, $\Sigma \vdash$ (INTERSECTS s w).

Note that the singleton type $\{A\text{-}Thing\}$ is always Σ-inhabitable. If s is a class expression such that $\Sigma \vdash$ (THERE-EXISTS s), then s^* is defined to be the Υ-type consisting of all class expressions w such that $\Sigma \vdash$ (IS s w). Inference rule 5 guarantees that s^* contains s and inference rules 2, 12, 4, and 15 guarantee that s^* satisfies the four requirements respectively in the definition of a Σ-inhabitable Υ-type and thus s^* is always Σ-inhabitable. If s is a class expression (possibly outside of Υ) such that $\Sigma \not\vdash$ (THERE-EXISTS s) then s^* is defined to be the singleton type $\{A\text{-}Thing\}$.

A class expression s in Υ will be called Σ-atomic if $\Sigma \vdash$ (THERE-EXISTS s) and $\Sigma \vdash$ (DETERMINED s). Note that if s is a Σ-atomic class expression then the type s^* contains s. Furthermore, one can show that if s is Σ-atomic then s^* is the *only* Σ-inhabitable Υ-type that contains s. More specifically, consider a Σ-inhabitable type τ that contains s. The definition of Σ-inhabitability ensures that s^* is a subset of τ. To show that τ is a subset of s^*, consider a class expression t in τ. The definition of Σ-inhabitable ensures that $\Sigma \vdash$ (INTERSECTS s t). But inference rule 20 then ensures that $\Sigma \vdash$ (IS s t) and thus t is an element of s^*. Thus any Σ-atomic class expression in Υ is contained in exactly one Σ-inhabitable Υ-type. An Υ-type τ will be called Σ-atomic if it is of the form s^* for some Σ-atomic class expression s. Note that a Σ-inhabitable type τ is Σ-atomic if and only if τ contains a class expression s such that $\Sigma \vdash$ (DETERMINED s), in which case τ equals s^*.

It is tempting to define the semantic domain of the desired model of Σ to be the set of Σ-inhabitable types. Unfortunately, this does not allow for infinite domains and Σ may not have finite models. The need for infinite domains arises from the need to include "predecessors". If the type τ contains a class expression of the form $f(s)$ where f is a function symbol, then any

domain element d that inhabits the type τ must be a member of the class denoted by $f(s)$ and thus there must be some predecessor domain element d' in the class denoted by s such that $f(d')$ equals d. If $\Sigma \vdash$ (IS s $f(s)$) then the need to include a predecessor for each element of s may force an infinite domain.

An infinite domain can be constructed by taking the domain elements to be pairs of the form $<\tau, \alpha>$ where τ is a Σ-inhabitable Υ-type and α is an expression that specifies the role played by the domain element. More specifically, the domain D is defined inductively as follows. Every Σ-inhabitable type must have at least one inhabitant in the model. Thus for, every Σ-inhabitable type τ, D contains the pair $<\tau, 0>$. If τ is Σ-inhabitable but not Σ-atomic then we require that D contain at least two inhabitants of τ; we specify that D contains the pair $<\tau, 1>$ as well as the pair $<\tau, 0>$. Finally, if D contains the pair $<\tau, \alpha>$, and τ contains a class expression of the form $f(s_1, \ldots s_n)$, where some s_i^* is not Σ-atomic, then D contains the "predecessor" pair $<s_i^*, f(s_1, \ldots s_n) \mapsto <\tau, \alpha>>$ where s_i is the first class expression among $s_1, \ldots s_n$ such that s_i^* is not Σ-atomic.

There are several things worth noting about the semantic domain D. First, note that if τ is a Σ-atomic type then the definition of D directly guarantees that $<\tau, 0>$ is the only pair in D whose first component is the type τ. Second, note that all elements of D are either of the form $<\tau, 0>$, $<\tau, 1>$ of $<s_j^*, f(s_1, \ldots s_n) \mapsto <\tau, \alpha>>$ where $f(s_1, \ldots s_n)$ is a member of the type τ. Finally, note that D can be infinite. More specifically, if s is a class expression in Υ such that $\Sigma \vdash$ (THERE-EXISTS s), but s is not Σ-atomic, and $\Sigma \vdash$ (IS s $f(s)$) for some function symbol f, then for each pair $<s^*, \alpha>$ in the semantic domain D, the domain D will contain a "predecessor" pair $<s^*, f(s^*) \mapsto <s^*, \alpha>>$.

To complete the specification of the model of Σ we must give the interpretation of the constant, function, and predicate symbols. A constant c is interpreted to be the pair $<c^*, 0>$. A monadic predicate symbol P is interpreted to be the set of all pairs $<\tau, \alpha>$ where the type τ contains the symbol P. A k-ary predicate symbol R is interpreted as the set of tuples $<<s_1^*, 0>, \ldots <s_{k-1}^*, 0>, <\tau, \alpha>>$ such that τ contains the class expression $R(s_1, \ldots s_{k-1})$. Finally, consider applying the function denoted by the symbol

f to the arguments $<<\tau_1, \alpha_1>, \ldots <\tau_k, \alpha_k>>$. We will say that a particular argument $<\tau_i, \alpha_i>$ determines f on the arguments $<<\tau_1, \alpha_1>, \ldots <\tau_k, \alpha_k>>$ if $<\tau_i, \alpha_i>$ is of the form $<s_i^*, f(s_1, \ldots s_n) \mapsto <\sigma, \beta>>$ where for each s_j, the type s_j^* equals the type τ_j. The definition of D implies that if $<\tau_i, \alpha_i>$ determines f on $<<\tau_1, \alpha_1>, \ldots <\tau_k, \alpha_k>>$, then τ_i, which is equal to s_i^*, must be the first type in the sequence $\tau_1, \ldots \tau_n$ that is not Σ-atomic. This implies that there can be at most one argument $<\tau_i, \alpha_i>$ that determines f on the tuple $<<\tau_1, \alpha_1>, \ldots <\tau_k, \alpha_k>>$. If such an argument exists, we define the value of f on this tuple of arguments to be the pair $<\sigma, \beta>$ given by the distinguished argument. If there does not exist such an argument, then the value of f on these pairs equals $<\sigma, 0>$ where σ is the union of all types of the form $f(s_1, \ldots s_k)^*$ where each s_j is a member of the type τ_j. The rules of obviousness for intersection-formulas ensure that σ is a Σ-inhabitable Υ-type.

Given the rules of inference listed in section 3 it is possible to prove that under this semantic interpretation the Υ-type of a pair $<\tau, \alpha>$ is, in fact, the type τ. This is equivalent to the statement that for any class expression s in Υ, the set denoted by s under this interpretation contains a pair $<\tau, \alpha>$ if and only if τ contains s. This latter statement can be proven by structural induction on the class expression s. Consider a constant symbol c in Υ. First we show that if $<\tau, \alpha>$ is in the class denoted by c then c is a member of τ. The constant c denotes the singleton class containing the pair $<c^*, 0>$. Inference rule 6 guarantees that $\Sigma \vdash$ (THERE-EXISTS c) and therefore c^* contains c. Next we suppose that c is a member of a Σ-inhabitable type τ and show that any pair of the form $<\tau, \alpha>$ is contained in the class denoted by c. Inference rules 6 and 7 gaurantee that c is Σ-atomic and therefore c^* is the only Σ-inhabitable type that contains c. Thus τ must be c^*. Furthermore, the type c^* is Σ-atomic and therefore the only domain element whose first component is c^* is the pair $<c^*, 0>$. Now consider a monadic class expression P in Υ. The class denoted by P is the set of pairs $<\tau, \alpha>$ such that τ contains P so the result follows by definition. Applications of relations and functions are somewhat more complex. For application class expressions the result is proven using properties provided by the inference rules together with the assumption that the statement holds on the subexpressions of the application in question. Most of the cases are not given here, but one particular case is worth noting. Suppose that $f(s_1, \ldots s_n)$ is a member of the type σ. In this case we must show that all pairs of the

form $<\sigma, \alpha>$ are members of the class denoted by $f(s_1, \ldots s_n)$. There are two subcases. First, suppose that each class expression s_1 is Σ-atomic. Since each s_i is Σ-atomic, inference rules 8 and 9 guarantee that $f(s_1, \ldots s_n)$ is also Σ-atomic. Since $f(s_1, \ldots s_n)$ is both Σ-atomic and a member of the type σ, σ must be the type $f(s_1, \ldots s_n)^*$ and α must be 0. To show that $<\sigma, \alpha>$ is a member of the class denoted by $f(s_1, \ldots s_n)$ it now suffices to show that the class denoted by $f(s_1, \ldots s_n)$ contains the pair $<f(s_1, \ldots s_n)^*, 0>$. Since each s_i is Σ-atomic, each expression s_i denotes the class containing the single pair $<s_i^*, 0>$. The semantic definition of f specifies that in this case the value of the expression $f(s_1, \ldots s_n)$ is the pair $<\tau, 0>$ where τ is the union of all types of the form $f(t_1, \ldots t_n)$ where t_i is a member of s_i^* for each t_i. But the type $f(s_1, \ldots s_n)^*$ is included in this union and thus τ contains the class $f(s_1, \ldots s_n)^*$. Since $f(s_1, \ldots s_n)$ is Σ-atomic, this implies that τ equals $f(s_1, \ldots s_n)^*$ so the result holds. Returning to the second subcase, suppose that $f(s_1, \ldots s_n)$ is a member of σ but that there exists some s_i that is not Σ-atomic. Let s_i be the first such such non-Σ-atomic argument. The definition of D guarantees that D contains a pair $<s_i^*, f(s_1, \ldots s_n) \mapsto <\sigma, \alpha>>$. Since σ contains $f(s_1, \ldots s_n)$, $\Sigma \vdash$ (THERE-EXISTS $f(s_1, \ldots s_n)$). Inference rule 11 guarantees that $\Sigma \vdash$ (THERE-EXISTS s_j) for each s_j. Therefore s_j^* contains s_j for each type s_j^*. By the induction hypothesis, the pairs $<s_1^*, 0>, \ldots$ $<s_i^*, f(s_1, \ldots s_n) \mapsto <\sigma, \alpha>>, \ldots <s_n^*, 0>$ are members of the classes denoted by $s_1, \ldots s_n$ respectively. But the semantic interpretation of the function f guarantees that f applied to these arguments yields the pair $<\sigma, \alpha>$ and thus the pair $<\sigma, \alpha>$ is a member of the class denoted by $f(s_1, \ldots s_n)$.

Given that the Υ-type of $<\tau, \alpha>$ is the type τ, i.e., that $<\tau, \alpha>$ is a member of the class denoted by s if and only if s is a member of τ, the definition of a Σ-inhabitable Υ-type implies several "default properties" of the semantic interpretation. More specifically, for any class expression s in Υ, if $\Sigma \not\vdash$ (THERE-EXISTS s) then s denotes the empty set. Similarly, for any two class expressions s and t in Υ, if $\Sigma \not\vdash$ (INTERSECTS s t) then the sets denoted by s and t are disjoint. Finally, if s is a class expression such that $\Sigma \vdash$ (THERE-EXISTS s), then if $\Sigma \not\vdash$ (DETERMINED s) then s denotes a set with more than one element, and if t is in Υ and $\Sigma \not\vdash$ (IS s t) then the set denoted by s is not a subset of the set denoted by t. These default properties, together with inference rules 10 and 14, ensure that this semantic interpretation is a model of Σ.

6 Extended Rules of Obviousness

To compare taxonomic and classical syntax more directly, we define two high-level proof systems: one based on classical syntax and one based on taxonomic syntax. The system based on taxonomic syntax is constructed from a modification of the decision procedure discussed in section 4. Section 8 defines the high-level proof system based on taxonomic syntax. Given the specification for the taxonomic high-level proof system, the adaptation of that system to classical syntax is presented in section 10.

The first step in defining the high-level proof system is to define a technical notion of an obviously true statement. The obviously true statements are defined by certain rules of obviousness. Each rule of obviousness states that if certain antecedent facts are obvious then a certain conclusion is also obvious. The rules of obviousness contain many, but not all, of the inference rules needed for a complete inference system for first order taxonomic formulas. The rules of obviousness include all of the rules of section 3 together with certain additional rules specified in this section. These additional rules involve a set of variables $\mathcal{F}$ called the *focus set.* We write $\Sigma, \mathcal{F} \vdash\!\circ\ \Psi$ if there exists a derivation of Ψ from the formulas in Σ using the extended rules of obviousness with focus set $\mathcal{F}$. The notation $\Sigma, \mathcal{F} \vdash\!\circ\ \mathbf{F}$ is analogous to the notation $\Sigma \vdash\!\circ\ \mathbf{F}$ used above.

In taxonomic syntax there are no explicit quantifiers in formulas; all taxonomic formulas are either atomic formulas or Boolean combinations of atomic formulas. Since there are no quantified formulas, no rules of obviousness are needed for quantified formulas. Class expressions, on the other hand, can involve quantifiers. Figure 2 gives rules of obviousness for such-that class expressions. Intuitively, the rules of obviousness for such-that expressions only allow the such-that quantifier to be instantiated with focus objects. The restriction of the instantiation of quantifiers to focus objects makes it possible to write a procedure for determining obviousness.

Rule 24 can be derived from rules 22 and 23. For example, suppose y and z are focus objects such that one can derive (IS z y) and $\Phi(y)$. In this case rule 22 allows one to derive (IS y (*A-Thing* x S.T. $\Phi(x)$)). By transitivity one can derive (IS z (*A-Thing* x S.T. $\Phi(x)$)). Finally, by rule 23 one can derive

(21) $(\texttt{IS}\ (s\ x\ \texttt{S.T.}\ \Phi(x))\ s)$

(22) $\dfrac{(\texttt{IS}\ y\ s),\ \Phi(y)}{(\texttt{IS}\ y\ (s\ x\ \texttt{S.T.}\ \Phi(x)))}$

(23) $\dfrac{(\texttt{IS}\ y\ (s\ x\ \texttt{S.T.}\ \Phi(x)))}{\Phi(y)}$

(24) $\dfrac{(\texttt{IS}\ z_1\ y_1)\ldots(\texttt{IS}\ z_n\ y_n),\ \Phi(y_1\ldots y_n)}{\Phi(z_1,\ldots z_n)}$

Figure 2: The inference rule for such-that class expressions. The variables y, y_i, and z_i must be members of the focus set $\mathcal{F}$.[4]

$\Phi(z)$. Thus, it would appear that rule 24 is unnecessary. However, rule 24 is needed in constructing a decision procedure for the extended rules. More specifically, the decision procedure uses label propagation on a finite network. Rule 24 allows certain inferences on the finite network that would otherwise not be performed unless the network were extended to include additional such-that class expressions.

In addition to the above rules for such-that expressions, the extended rules of obviousness include rules for Boolean connectives. We assume that all Boolean formulas are constructed using the connectives OR and NOT. The rules of obviousness for Boolean formulas are listed in figure 3.

Inference rules 25 through 31 are not complete for Boolean inference. For example, rules 25 through 31 cannot be used to deduce Ψ from (OR Φ Ψ) and (OR (NOT Φ) Ψ). Intuitively, the rules do not allow for case analysis. The rules are designed so that the inference relation generated by the rules is both reasonably powerful and quickly decidable.

Note that rule 31 can be derived from rules 25 and 28. More specifically, suppose that one can derive both Ψ and (NOT Ψ). In this case, rule 25 allows one to derive (OR Ψ Φ). Rule 28 then allows one to derive Φ. Thus it would

[4]We use the notation $\Phi(y_1,\ldots y_n)$ as an abbreviation for $\Phi[y_1/w_1,\ldots y_n/w_n]$, i.e., the simultaneous substitution of y_i for all free occurences of w_i in the expression Φ with appropriate renaming of bound variables. Note that in rule (24) both y_i and w_i may occur free in Φ and so the expression $\Phi(z_1,\ldots z_n)$ may include y_i as a free variable.

(25) $\dfrac{(\texttt{OR}\ \Phi\ \Psi),\ (\texttt{NOT}\ \Phi)}{\Psi}$

$\dfrac{(\texttt{OR}\ \Phi\ \Psi),\ (\texttt{NOT}\ \Psi)}{\Phi}$

(26) $\dfrac{(\texttt{NOT}\ (\texttt{OR}\ \Phi\ \Psi))}{(\texttt{NOT}\ \Phi)}$

$\dfrac{(\texttt{NOT}\ (\texttt{OR}\ \Phi\ \Psi))}{(\texttt{NOT}\ \Psi)}$

(27) $\dfrac{(\texttt{NOT}\ \Phi),(\texttt{NOT}\ \Psi)}{(\texttt{NOT}\ (\texttt{OR}\ \Phi\ \Psi))}$

(28) $\dfrac{\Phi}{(\texttt{OR}\ \Phi\ \Psi)}$

$\dfrac{\Psi}{(\texttt{OR}\ \Phi\ \Psi)}$

(29) $\dfrac{(\texttt{NOT}\ (\texttt{NOT}\ \Phi))}{\Phi}$

(30) $\dfrac{\Phi}{(\texttt{NOT}\ (\texttt{NOT}\ \Phi))}$

(31) $\dfrac{\Psi \quad (\texttt{NOT}\ \Psi)}{\Phi}$

Figure 3: Rules of Obviousness for Boolean formulas.

appear that rule 31 is not needed. However, the decision procedure for the inference relation is implemented as label propagation on a finite network. Inference rule 31 allows for the derivation of labels that would not otherwise be derivable unless the network were expanded to include certain disjunctions. In practice, of course, the propagation process can be terminated whenever a contradiction is discovered.

Before giving a label-propagation decision procedure for these rules, some additional terminology is needed. In the following definitions Σ is taken to be a fixed but arbitrary set of formulas, $\mathcal{F}$ is a fixed but arbitrary set of variables (called focus objects), and Ψ is a fixed but arbitrary formula.

> **Definition:** An *extended label formula* for a set Υ of expressions is either a formula that is a member of Υ, the negation of a formula that is a member of Υ, or a formula of the form

(THERE-EXISTS s), (DETERMINED s), (IS s t), or (INTERSECTS s t) where s and t are class expressions that are members of Υ.

Definition: A set of expressions Υ (containing both class expressions and formulas) is said to be *closed over* Σ, $\mathcal{F}$ and Ψ if

- Υ contains *A-Thing*,
- Υ contains Ψ plus every member of Σ and $\mathcal{F}$,
- every subexpression of every member of Υ is also a member of Υ,
- and for every such-that class expression (s x S.T. $\Phi(x)$) in Υ, and every variable y in $\mathcal{F}$, the formula $\Phi(y)$ is also in Υ.

Definition: For any set of expressions Υ we write $\Sigma, \mathcal{F} \vdash_{\Upsilon} \Psi$ if there exists a derivation of Ψ using the extended rules of obviousness such that every formula in that derivation is an extended label formula of Υ.

Definition: We write $\Sigma, \mathcal{F} \vdash \Psi$ if $\Sigma, \mathcal{F} \vdash_{\Upsilon} \Psi$ where Υ is the least set of expressions closed over Σ, $\mathcal{F}$, and Ψ.

It is possible to show that, as long as Σ and $\mathcal{F}$ are finite, the least set of expressions closed over Σ, $\mathcal{F}$ and Ψ is also finite. More specifically, the number of expressions in the least set closed over Σ, $\mathcal{F}$, and Ψ is no larger than $1 + |\mathcal{F}| + [\Gamma] + [\Gamma]|\mathcal{F}|^Q$ where Γ is the set $\Sigma \cup \{\Psi\}$, $[\Gamma]$ is the number of expressions that are either members of Γ or appear in members of Γ, $|\mathcal{F}|$ is the number of elements of $\mathcal{F}$, and Q is maximum level of quantifier nesting that appears in Γ. In practice the level of quantifier nesting remains small (three or four) and the size of the least set closed over Σ, $\mathcal{F}$ and Ψ is usually much smaller than this worst-case bound. Note that if an upper bound is placed on both the number of focus objects and the maximum level of quantifier nesting, then the size of the least set closed over Σ, $\mathcal{F}$ and Ψ remains linear in the size of $\Sigma \cup \{\Psi\}$.

For any finite set Υ one can determine whether or not $\Sigma, \mathcal{F} \vdash_{\Upsilon} \Psi$ using a label propagation procedure on a network representing the set Υ. Unlike the

network described in section 4, the network used for the extended rules of inference contains nodes that represent formulas as well as nodes that represent class expressions. A data structure that represents a formula must be updated whenever that formula is derived using the rules of obviousness, and updated in a different way whenever the negation of the formula is derived. An analysis similar to that given in section 4 shows that the propagation process can be implemented in a way that requires at most $O(n^3)$ time where n is the number of expressions in Υ, assuming that hash table lookups take constant time. As discussed in section 4, there is no loss of generality in assuming that applications involve at most two arguments.

We have not yet ruled out the possibility that the unbounded inference relation $\vdash_O$ may be more powerful than the inference relation $\vdash$ defined by the bounded label-propagation mechanism, i.e., it seems possible that $\Sigma, \mathcal{F} \vdash_O \Psi$ and yet $\Sigma, \mathcal{F} \not\vdash \Psi$. It turns out, however, that the bounded relation is as powerful as the unbounded relation and thus the decision procedure for the bounded relation is also a decision procedure for the unbounded relation. The proof of this fact is presented in the following section.

7 Correctness of the Extended Decision Procedure

The claim that for finite Σ and $\mathcal{F}$ one can determine whether or not $\Sigma, \mathcal{F} \vdash_O \Psi$ rests on the claim that the relation $\vdash_O$ is the same as the restricted relation $\vdash$. Since both of these relations are semantically sound, and $\vdash$ is clearly a subrelation of $\vdash_O$, it would be sufficient to show that $\vdash$ is semantically complete. Unfortunately, neither $\vdash$ nor $\vdash_O$ are semantically complete — the semantic entailment relation for full taxonomic syntax is undecidable. Since no purely semantic proof is possible, we give a syntactic proof that that $\vdash$ is the same as $\vdash_O$.

Suppose that $\Sigma, \mathcal{F} \not\vdash \Psi$. By the definition of $\vdash$, this implies that $\Sigma, \mathcal{F} \not\vdash_\Upsilon \Psi$ where Υ is the least set closed over Σ, $\mathcal{F}$ and Ψ. To prove that $\Sigma, \mathcal{F} \not\vdash_O \Psi$, it suffices to prove that $\Sigma, \mathcal{F} \not\vdash_{\Upsilon'} \Psi$ for any finite extension Υ' of Υ. This can be established by expanding Υ one expression at a time.

Definition: A *one step $\mathcal{F}$-extension* of a set Υ is an expression α that is either

- a monadic predicate symbol,
- a constant symbol,
- a variable,
- an atomic formula that is a label formula of Υ,
- the negation of a formula in Υ,
- a disjunction of two formulas in Υ,
- an application $R(s_1, \ldots s_n)$ where R is either a relation or function symbol and each s_i is a class expression in Υ,
- a such-that expression $(s\ x\ \texttt{S.T.}\ \Phi(x))$ where s, x, and $\Phi(x)$ are all members of Υ and for each y in $\mathcal{F}$, $\Phi(y)$ is a member of Υ.

If Υ is closed over Σ, $\mathcal{F}$ and Ψ, and α is a one step $\mathcal{F}$-extension of Υ, then $\Upsilon \cup \{\alpha\}$ is also closed over Σ, $\mathcal{F}$, and Ψ. Furthermore, as long as the focus set $\mathcal{F}$ is finite, the set Υ can be extended by a series of one step $\mathcal{F}$-extensions to include any desired expression.[5] Thus, it suffices to prove that if $\Sigma, \mathcal{F} \not\vdash_\Upsilon \Psi$ where Υ is closed over Σ, $\mathcal{F}$, and Ψ, and α is any one step $\mathcal{F}$-extension of Υ, then $\Sigma, \mathcal{F} \not\vdash_{\Upsilon \cup \{\alpha\}} \Psi$.

Now consider an arbitriary set Υ that is closed over Σ, $\mathcal{F}$, and Ψ such that $\Sigma, \mathcal{F} \not\vdash_\Upsilon \Psi$, let α be a one step $\mathcal{F}$-extension of Υ, and let Υ' be the set $\Upsilon \cup \{\alpha\}$. We must prove that $\Sigma, \mathcal{F} \not\vdash_{\Upsilon'} \Psi$. For the purposes of this proof we define a *new label formula* to be an extended label formula of Υ' that is not an extended label formula of Υ. The label formulas of Υ will be called *old* label formulas. We say that an old label formula Θ *was already derivable* if $\Sigma, \mathcal{F} \vdash_\Upsilon \Theta$. We say that an extended label formula Θ of Υ', either new or old, is *newly derived* if Θ was not already derivable and $\Sigma, \mathcal{F} \not\vdash_\Upsilon \Theta$. Since Υ is closed over Σ, $\mathcal{F}$, and Ψ, the formula Ψ must be a member of Υ and thus Ψ

[5] Although we are only interested in the case where $\mathcal{F}$ is finite, the relations $\vdash\!\!\circ$ and $\vdash$ are well defined for infinite focus sets. One can prove that even for infinite focus sets these two relations are the same. If $\mathcal{F}$ is infinite, one must consider transfinite sequences of one step $\mathcal{F}$-extensions.

is an old label formula. To show that $\Sigma, \mathcal{F} \not\vdash_{\Upsilon} \Psi$, it suffices to prove that no old label formula is newly derived, or equivalently, that every newly derived formula is a new label formula.

In proving that every newly derived formula is a new label formula we can assume that α is not a member of Υ and that we cannot derive a contradiction by label propagation on Υ, i.e., there is no Φ such that both Φ and (NOT Φ) were already derived (if a contradiciton is already derivable then all old label formulas are also already derivable). Consider the kinds of expression that α might be. If α is a monadic predicate symbol then an examination of the inference rules shows that the only newly derived formula is (IS α α). If α is a constant symbol or a variable then a similar examination of the inference rules shows that the only newly derived formulas are (IS α α), (THERE-EXISTS α), (DETERMINED α), and (INTERSECTS α α). The other cases are more complex.

Suppose α is an atomic formula that is a label formula of Υ. In this case the formula (NOT α) becomes a new label formula. In fact, it is the only new label formula. None of rules 1 through 22 can derive a non-atomic formula and thus none of these rules can be used to derive (NOT α). To see that rule 23 cannot derive (NOT α) note that since Υ is closed over over Σ, $\mathcal{F}$, and Ψ, for any such-that class expression (s x S.T. $\Phi(x)$) in Υ and any y in $\mathcal{F}$ the formula $\Phi(y)$ must be in Υ and thus $\Phi(y)$ cannot be the new label formula (NOT α). Skipping over rule 24 for the moment, we note that rules 25, 26 and 29 fail to derive (NOT α) because (NOT α) is not contained in any Boolean label formulas. Rules 27, 28, and 30 cannot derive negations. Finally, rule 31 does not apply because, by assumption, no contradiction can be derived by label propagation on Υ. Thus, the only way of deriving (NOT α) is with inference rule 24. In this case α must be of the form $\Phi(y)$ where y is a member of $\mathcal{F}$ and there must exist some z in $\mathcal{F}$ such (IS y z) and (NOT $\Phi(z)$) were already derivable. We must show that if (NOT α) is derived with inference rule 24 then no old label formulas can be newly derived. A syntactic analysis of the rules, using the observation that α does not appear as a proper subformula of any formulas in Υ', shows that the only inference rules that can use (NOT α) as a premise are inference rules 10, 14, 24 and 31. The only way inference rule 31 could apply is if the formula $\Phi(y)$ was already derivable. In this case inference rule 24 ensures that $\Phi(z)$ was already derivable. But this violates the assumption that no contradiction was already derivable. If some instance

of rule 10 or 14 can be used to derive an old label formula $\Theta(y)$ from the premise (NOT $\Phi(y)$), then the formula $\Theta(z)$ must already have been derivable by the same rule. In this case the formula $\Theta(y)$ must already have bee.. derivable from $\Theta(z)$ by inference rule 24.

The cases where α is either the negation of a member of Υ or the disjunction of two members of Υ are similar to the case where α is an atomic label formula and will not be discussed in detail here. It remains only to consider the two cases where α is a class expression other than a constant or monadic predicate symbol. Suppose that α is an application $R(s_1, \ldots s_n)$ where each class expression s_i is a member of Υ. In this case the new label formulas are all atomic formulas involving the class α. We wish to show that all of the newly derived formulas are new label formulas. To show this we show that the inference rules maintain the following invariants:

- Every newly derived formula is a new label formula.
- If (IS α t) is newly derived where t is in Υ, then either (IS *A-Thing* t) was already derived, or there exists a class expression $R(w_1, \ldots w_n)$ in Υ such that (IS s_i w_i) was already derived for each w_i and the formula (IS $R(w_1, \ldots w_n)$ t) was also already derived.
- If (IS t α) is newly derived where t is in Υ, then either:
  1. Υ contains a class expression $R(w_1, \ldots w_n)$ such that (IS w_i s_i) was already derived for each w_i and (IS t $R(w_1, \ldots w_n)$) was also already derived, or
  2. there exists a class expression t' in Υ such that (DETERMINED t') and (IS t t') were already derived, and (INTERSECTS t' α) will be newly derived.
- If (THERE-EXISTS α) or (INTERSECTS α α) is newly derived, then either R is a function symbol and (THERE-EXISTS s_i) was already derived for each s_i, or there exists some members t and $R(w_1, \ldots w_n)$ of Υ such that (THERE-EXISTS t) and(IS t $R(w_1, \ldots w_n)$) were already derived, and (IS $R(w_1, \ldots w_n)$ α) will be newly derived.

- If (DETERMINED α) is newly derived, then either R is a function symbol and (DETERMINED s_i) was already derived for each s_i or there exists some member t of Υ such that (IS α t) will be newly derived and (DETERMINED t) was already derived.

- If (INTERSECTS α t) or (INTERSECTS t α) is newly derived, where t is in Υ, then either (IS *A-Thing* t) was already derived and (THERE-EXISTS α) will be derived, or there exists a class expression $R(w_1, \ldots w_n)$ in Υ such that either:

  1. R is a function symbol, and the formulas (INTERSECTS w_1 s_1), ... (INTERSECTS w_n s_n), and (IS $R(w_1, \ldots w_n)$ t) were already derived, or
  2. formulas (IS w_1 s_1), ... (IS w_n s_n), and (INTERSECTS $R(w_1, \ldots w_n)$ t) were already derived.

Since all new label formulas are atomic formulas not contained in any Boolean formulas in Υ, none of the Boolean rules apply (i.e., none of them can fire as long no old label formulas are newly derived). The definition of closure over Σ, $\mathcal{F}$, and Ψ ensure that rules 21, 22, and 23 do not apply. Thus we need only check these invariants for inference rules 24 and 1 though 20. We will spare the reader the laborious case analysis necessary to verify that these rules maintain the above invariants.

Now suppose that α is a such-that class expression (s x S.T. $\Phi(x)$). This case is similar to the case where α is an application; we show that the inference rules preserve a certain set of invariants. To state the invariants that are preserved in this particular case we first define an *α-witness* to be an element y of the focus set $\mathcal{F}$ such that $\Sigma, \mathcal{F} \vdash_\Upsilon$(IS y s) and $\Sigma, \mathcal{F} \vdash_\Upsilon \Phi(y)$. For any α-witness y inference rule 20 gurantess that the formula (IS y α) will be newly derived. Given the notion of an α-witness, the invariants maintained by the inference rules can be concisely stated as follows:

- Every newly derived formula is a new label formula.

- If (IS α t) is newly derived and t is in Υ then (IS s t) was already derived.

- If (IS t α) is newly derived and t is in Υ then there exists an α-witness y such that (IS t y) was already derived.
- If (THERE-EXISTS α) or (INTERSECTS α α) is newly derived then there exists an α-witness.
- If (DETERMINED α) is newly derived then (DETERMINED s) was already derived.
- If (INTERSECTS t α) or (INTERSECTS α t) is newly derived and t is a member of Υ then there exists an α-witness y such that (IS y t) was already derived.

As in the previous case, all of the new label formulas are atomic formulas that do not appear in any Boolean expressions that are members of Υ. This implies that none of the Boolean rules apply. We again spare the reader the laborious case analysis necessary to verify that rules 1 through 24 preserve the above invariants.

This completes our presentation of the proof that $\vdash$ is the same as $\vdash_o$. This result can be summarized in the statement that the restricted relation $\vdash$ is *syntactically complete* relative to the unrestricted relation $\vdash_o$. The proof involves a fairly long case analysis most of which has not been explicitly given here. This is unfortunate because many of the inference rules and definitions presented in this paper are motivated by the desire that $\vdash$ be syntactically complete relative to $\vdash_o$. The long case analysis required to prove the syntactic completeness of $\vdash$ obscures the role played by particular inference rules and definitions. In spite of considerable effort, we have not been able to find a more concise proof of the equivalence of $\vdash$ and $\vdash_o$.

8 A High-Level Proof System

A high-level proof is a series of lines where each line contains a "sequent" of the form $\Sigma \vdash \Phi$ where Σ is a set of formulas and Φ is either a formula or the special token **F**.[6]. The lines of a high-level proof are divided into two kinds:

[6] A more "user-friendly" syntax for high-level proofs is given in [McAllester, 1989]

syntactically derived lines and unjustified lines. A syntactically derived line is a line that can be derived from previous lines using one of the following five high-level proof rules. Each high-level proof rule is a form of universal generalization.[7] The need to include rules of universal generalization in the high-level proof system will be discussed further in the presentation of the high-level completeness proof (section 9). In the following rules x, and each x_i, must be a variable that does not appear free in any formula in Σ or in any of the class expressions s, t or s_i. In the last rule z must be a variable but there are no restrictions on where z can appear, e.g. z may appear free in Σ or any s_i.

$$\frac{\Sigma \vdash (\texttt{NOT}\ (\texttt{IS}\ x\ s))}{\Sigma \vdash (\texttt{NOT}\ (\texttt{THERE-EXISTS}\ s))}$$

$$\frac{\Sigma \cup \{(\texttt{IS}\ x_1\ s), (\texttt{IS}\ x_2\ s)\} \vdash (\texttt{IS}\ x_1\ x_2)}{\Sigma \vdash \{(\texttt{DETERMINED}\ s)\}}$$

$$\frac{\Sigma \cup \{(\texttt{IS}\ x\ s),\ (\texttt{IS}\ x\ t)\} \vdash \mathbf{F}}{\Sigma \vdash (\texttt{NOT}\ (\texttt{INTERSECTS}\ s\ t))}$$

$$\frac{\Sigma \cup \{(\texttt{IS}\ x\ s)\} \vdash (\texttt{IS}\ x\ t)}{\Sigma \vdash (\texttt{IS}\ s\ t)}$$

$$\frac{\Sigma \cup \{(\texttt{IS}\ x_1\ s_1), \ldots (\texttt{IS}\ x_n\ s_n)\} \vdash (\texttt{NOT}\ (\texttt{IS}\ z\ R(x_1, \ldots x_n)))}{\Sigma \vdash (\texttt{NOT}\ (\texttt{IS}\ z\ R(s_1, \ldots s_n)))}$$

A line of a high-level proof that is not derived from previous lines using one of the high-level generalization rules is called an *unjustified* line. Each

[7] In a user-friendly version of the high-level proof system, each high-level rule of universal generalization appears in its contrapositive form; rather than derive a universal statement from a statement about an arbitrary individual, the user-friendly high-level system allows one to introduce witnesses based on existential statements.

unjustified line in a high-level proof must be explicitly associated with a set of variables called the *focus set* of that line. Consider an unjustified line $\Sigma \vdash \Phi$ with associated focus set $\mathcal{F}$. Intuitively, each unjustified line must obviously follow from previous lines in the proof. Let Σ' be Σ plus all formulas previously proven to follow from Σ, i.e., all formulas Ψ such that the proof contains an earlier line of the form $\Gamma \vdash \Psi$ where Γ is a subset of Σ. An unjustified line $\Sigma \vdash \Phi$ with associated focus set $\mathcal{F}$ must follow from previous lines. More specifically, if Φ is the constant $\mathbf{F}$, then we must have $\Sigma', \mathcal{F} \vdash\!\circ\ \mathbf{F}$. If Φ is some formula other than $\mathbf{F}$, then we must have $\Sigma' \cup \{(\texttt{NOT}\ \Phi)\}, \mathcal{F} \vdash \mathbf{F}$.

It is important to be able to quickly determine if a series of high-level proof lines is acceptable, i.e. that each unjustified line satisfies the condition specified above. The cost of determining the acceptability of a given unjustified line is quite sensitive to the size of the focus set $\mathcal{F}$ associated with that line. The high-level completeness theorem given in the following section shows that if a formula Φ semantically follows from a set of formulas Σ then there exists a high-level derivation of the sequent $\Sigma \vdash \Phi$ such that each unjustified line involves at most one focus object. However, proofs can be made much shorter by allowing unjustified lines to be associated with more than one focus object. Thus there is a trade-off between proof length and the time required to machine verify the proof: short proofs, in which unjustified lines have many focus objects, take longer to machine verify than longer proofs in which unjustified lines are associated with fewer focus objects. In the proof of the Stone representation theorem from the axioms of set theory, described in [McAllester, 1989], unjustified lines involved up to ten focus objects. It is possible to show that the size of the network generated in determining if $\Gamma, \mathcal{F} \vdash\!\circ\ \mathbf{F}$ is, in the worst case, $O([\Gamma]|\mathcal{F}|^Q)$ where $[\Gamma]$ is the number of expressions that are either members of Γ or appear in some member of Γ, $|\mathcal{F}|$ is the number of elements of $\mathcal{F}$, and Q is the maximum level of quantifier-nesting that appears in any formula In Γ. In practice the maximum level of quantifier nesting remains small and, as a rule of thumb, the size of the network appears proportional to $[\Gamma]|\mathcal{F}|^3$.

9 High-Level Completeness

Throughout this section we only consider high-level proofs in which unjustified lines have at most one focus object. It turns out that this restricted high-level proof system is semantically complete for first order taxonomic formulas. More specifically, if a formula Φ semantically follows from a set of formulas Σ, then there exists a high-level proof that ends with the line $\Sigma \vdash \Phi$ and in which every unjustified line has at most one focus object. To prove this result one can first observe that there exists a high-level derivation of $\Sigma \vdash \Phi$ if and only if there exists a high-level derivation of $\Sigma \cup \{(\texttt{NOT}\ \Phi)\} \vdash \mathbf{F}$. To prove this it suffices to observe that, given a high-level derivation of $\Sigma \vdash \Phi$, the line $\Sigma \cup \{(\texttt{NOT}\ \Phi)\} \vdash \mathbf{F}$ can be immediately added as an unjustified line with an empty focus set. Similarly, given a derivation of $\Sigma \cup \{(\texttt{NOT}\ \Phi)\} \vdash \mathbf{F}$, the line $\Sigma \vdash \Phi$ can be acceptably added without justification. To prove the high-level system is complete, we assume that there is no derivation of $\Sigma \vdash \Phi$ and we show that in this case there exists a model of Σ in which Φ is false. If there is no derivation of $\Sigma \vdash \Phi$ then there must not be any high-level derivation of $\Sigma \cup \{(\texttt{NOT}\ \Phi)\} \vdash \mathbf{F}$. To prove that there exists a model of Σ in which Φ is false, it now suffices to show that, for any set of formulas Γ, if there is no derivation of $\Gamma \vdash \mathbf{F}$, then there exists some model of Γ.

Suppose that there is no derivation of $\Gamma \vdash \mathbf{F}$. One can construct a model of Γ using techniques analogous to those used in standard proofs of first order completeness. For simplicity we assume that the set of constant, function and predicate symbols in the language is countable and that there is a countably infinite set of variables. In this case one can enumerate all taxonomic formulas in an infinite sequence $\Theta_1, \Theta_2, \Theta_3 \ldots$.[8] Given that there is no derivation of $\Gamma \vdash \mathbf{F}$, one can then construct an infinite sequence of sets of formulas $\Omega_1, \Omega_2, \Omega_3 \ldots$ by setting Ω_1 equal to Γ and defining Ω_{j+1} as follows:

1. If there exists a derivation of $\Omega_j \vdash (\texttt{NOT}\ \Theta_j)$ then set Ω_{j+1} equal to Ω_j.

2. If there is no derivation of $\Omega_j \vdash (\texttt{NOT}\ \Theta_j)$, and Θ_j is a formula of the

[8]The completeness proof can be modified to handle uncountable languages, in which case one constructs a transfinite enumeration of formulas.

form (THERE-EXISTS s), then let x be some variable that does not appear in s or Ω_j and set Ω_{j+1} to be $\Omega_j \cup \{\Theta_j, (\text{IS } x\ s)\}$.

3. If there is no derivation of $\Omega_j \vdash (\text{NOT } \Theta_j)$, and Θ_j is a formula of the form (NOT (DETERMINED s)), then let x and y be variables that are not free in s or Ω_j and set Ω_{j+1} to be $\Omega_j \cup \{\Theta_j, (\text{IS } x\ s), (\text{IS } y\ s), (\text{NOT } (\text{IS } x\ y))\}$.

4. If there is no derivation of $\Omega_j \vdash (\text{NOT } \Theta_j)$, and Θ_j is a formula of the form (INTERSECTS s t), then let x be some variable that does not appear free in s, t or Ω_j and set Ω_{j+1} to be $\Omega_j \cup \{\Theta_j, (\text{IS } x\ s), (\text{IS } x\ t)\}$.

5. If there is no derivation of $\Omega_j \vdash (\text{NOT } \Theta_j)$, and Θ_j is a formula of the form (NOT (IS s t)) where s is not a variable, then let x be some variable that does not appear free in s, t or Ω_j and set Ω_{j+1} to be $\Omega_j \cup \{\Theta_j, (\text{IS } x\ s), (\text{NOT } (\text{IS } x\ t))\}$.

6. If there is no derivation of $\Omega_j \vdash (\text{NOT } \Theta_j)$ and Θ_j is a formula of the form (IS x $R(s_1, \ldots s_n)$) where x is a variable, then let $y_1, \ldots y_n$ be variables that do not appear free in Ω_j or in any of the class expressions s_j, and set Ω_{j+1} equal to $\Omega_j \cup \{\Theta_j, (\text{IS } x\ R(y_1, \ldots y_j)), (\text{IS } y_1\ s_1), \ldots (\text{IS } y_n\ s_n)\}$.

7. If none of the above conditions apply, then set Ω_{j+1} equal to $\Omega_j \cup \{\Theta_j\}$.

Given the high level proof rules introduced in the previous section, one can show that each Ω_j is a finite set of formulas that contains Γ and that there does not exist any derivation of $\Omega_j \vdash \mathbf{F}$. Steps 2, 3, 4, and 5 ensure that, if Θ_j is an existential statement that is a member of Ω_{j+1} then there are variables that act as witnesses to Θ_j in Ω_{j+1}. For example, if Θ_j is (THERE-EXISTS s) and Θ_j is a member of Ω_{j+1}, then there is some variable x such that Ω_{j+1} contains the formula (IS x s). Steps 2, 3, 4, 5, and 6 in the above specification directly correspond to the five high-level generalization rules presented in section 8. For each of these steps, the proof of the consistency of the newly constructed set Ω_{j+1} relies on the existence of the corresponding high-level generalization rule. Thus, the generalization rules in the high-level proof system are needed because they indirectly allow the introduction of witnesses for existential statements. In a user-friendly high-level proof system the high-level generalization rules can either be used directly or used in the contrapositive

form where they allow the introduction of new witnesses to previously proven existential statements.

Now let Ω be the union of all sets Ω_j. It is possible to show that Ω is both consistent and complete. More specifically, for any formula Ψ exactly one of the two formulas Ψ and (NOT Ψ) is contained in Ω. Furthermore, one can show that the set of formulas Ω is closed under all of the rules of obviousness where the rules for such-that expressions are no longer restricted to focus objects.

One can now define a first order structure whose domain consists of equivalence classes of variables. More specifically, for any variable x we define $|x|$ to be the set of variables y such that the formula (IS x y) is a member of Ω. The rules of obviousness for is-formulas ensure that these sets form equivalence classes of variables. We take the domain of the first order structure to be the collection of equivalence classes of the form $|x|$. It is now possible to define an interpretation of the variables, constants, functions, relations, and predicate symbols such that the semantic value of a class expression s equals the set of classes $|x|$ such that the formula (IS x s) is a member of Ω and such that, for every formula Ψ, the semantic interpretation makes Ψ true just in case Ψ is a member of Ω. This provides an interpretation of Γ. Thus one can establish that if there is no derivation of $\Gamma \vdash \mathbf{F}$ then there exists a semantic interpretation of Γ, and similarly, if there is no derivation of $\Sigma \vdash \Phi$, then there exists an interpretation of Σ in which Φ is false.

10 Taxonomic vs. Classical Syntax

To compare taxonomic and classical syntax we consider a high-level proof system analogous to the one defined in section 8 but based on classical rather than taxonomic syntax. A high-level proof in the system based on classical syntax is also a series of lines where each line is "sequent" $\Sigma \vdash \Phi$. Like the taxonomic system, the classical system is based on an obviousness relation $\vdash\!\circ$ and the high-level proof system allows unjustified lines where each unjustified line must be explicitly associated with a set of variables called the focus set for that line. The conditions under which an unjustified line is

acceptable are identical in both the taxonomic and classical systems except that the two systems are based on different obviousness relations. Although the obviousness relations underlying the two systems are different, each of the two obviousness relations is defined by a set of inference rules called rules of obviousness.

In the classical system the rules of obviousness presented in section 3 are replaced by the standard rules of inference for equality: reflexivity, symmetry, transitivity, and rules that allow the substitution of equals for equals in terms and atomic formulas. These rules of inference for equality are complete for classical literals: if the rules cannot derive a contradiction form a set of first order literals, then the set of literals is satisfiable.

The rules of obviousness that involve Boolean connectives are exactly the same in both the taxonomic and classical systems. In the classical system, we assume that the only quantifier is the classical universal quantifier $\forall$. The three taxonomic rules of obviousness involving such-that class expressions are replaced, in the classical system, by the following single rule of obviousness. In the following rule y must be a variable in $\mathcal{F}$.

$$\frac{\forall x \Phi(x)}{\Phi(y)}$$

The five high-level taxonomic generalization rules are replaced, in the classical system, by the following single high-level generalization rule. In the following rule x must be a variable that does not appear free in Σ.

$$\frac{\Sigma \vdash \Phi(x)}{\Sigma \vdash \forall x \Phi(x)}$$

Unlike taxonomic syntax, the classical rules of obviousness involving focus objects make the relationship between focus objects and previously proven lemmas explicit; the rules of obviousness allow any previously proven universal lemma to be applied to any focus object. In the taxonomic system, a formula of the form $\forall x \Phi(x)$ is represented by (IS *A-Thing* (*A-Thing* x S.T. $\Phi(x)$)).

If y is a focus object then the taxonomic rules of obviousness allow the derivation of (IS y *A-Thing*) and given the above is-formula, one can derive (IS y (*A-Thing* x S.T. $\Phi(x)$)). The rules of obviousness for such-that expressions then allow the derivation of $\Phi(y)$. Thus, the above classical rule of universal instantiation for focus objects is subsumed by the taxonomic rules of obviousness. In fact, all of the methods of deriving new lines in the classical high-level proof system are subsumed by methods of deriving new lines in the taxonomic high-level proof system. This claim can be formalized by giving a procedure for translating any proof in the classical high-level system into a corresponding proof in the taxonomic system.

For any classical first order formula Φ, the *taxonomic translation*, $T(\Phi)$ of the formula Φ is defined by structural induction on Φ. If Φ is an atomic formula of the form $R(s_1, \ldots s_n)$ then $T(\Phi)$ is the atomic taxonomic formula (IS s_n $R(s_1, \ldots s_{n-1})$). $T((\text{OR } \Theta\ \Psi))$ equals (OR $T(\Theta)$ $T(\Psi)$) and $T((\text{NOT } \Psi))$ equals (NOT $T(\Psi)$). If Φ is a universal formula $\forall x \Psi(x)$, then $T(\Phi)$ is the formula (IS *A-Thing* (*A-Thing* x S.T. $T(\Psi(x))$)). For any set Σ of classical first order formulas, $T(\Sigma)$ is the set of taxonomic formulas of the form $T(\Psi)$ for some Ψ in Σ. If P is a high-level proof in the classical high-level proof system, then $T(P)$ is the sequence of lines derived by translating each unjustified line $\Sigma \vdash \Phi$ in P to an unjustified line $T(\Sigma) \vdash T(\Phi)$ leaving the focus set of the line unchanged, and translating each universal generalization line $\Sigma \vdash \forall x \Phi(x)$ to an unjustified line of the form $T(\Sigma) \cup \{(\text{IS } x \textit{ A-Thing})\} \vdash$ (IS x (*A-Thing* x S.T. $\Phi(x)$)) with focus set $\{x\}$ followed by the generalization line $T(\Sigma) \vdash T(\forall x \Phi(x))$.

> **Taxonomic Domination Theorem:** The taxonomic proof system dominates the classical proof system in the sense that for any acceptable high-level proof P in the classical system, the proof $T(P)$ is acceptable in the taxonomic system.

Intuitively, the proof rules of the taxonomic system include the proof rules of the classical system as a special case. This is not a surprising result and is not difficult to prove. We conjecture, however, that the converse of this theorem does not hold, i.e., the taxonomic high-level proof system is *not* subsumed by the classical high-level proof system.

Strict Domination Conjecture: For any (large) constant k there exists a classical first order formula Φ and a taxonomic proof P of $T(\Phi)$ such that the shortest proof of Φ in the classical high-level proof system has length greater than k times the length of P.

If this conjecture is true, then there would exist a first order statement and a taxonomic proof of that statement such that the shortest classical proof is, say, a hundred times longer than the taxonomic proof.

11 Conclusion

We have defined a taxonomic syntax for first order predicate calculus and have presented several technical results describing computational properties of this syntax. Quantifier-free taxonomic literals are more expressive than literals of classical first order logic and yet there exists a polynomial time decision procedure for determining the satisfiability of a set of quantifier-free taxonomic literals. We have also investigated the value of taxonomic syntax in general theorem proving. We have define high-level proof systems for both taxonomic and classical systems and shown that the taxonomic system subsumes the classical system. Furthermore, we conjecture that the reverse is not true, i.e., that there exist high-level taxonomic proofs such that any classical high-level proof of the same result is much longer.

References

[Bledsoe, 1977] W. W. Bledsoe. Non-resolution theorem proving. *Artificial Intelligence*, 9:1–35, 1977.

[Bobrow and Winograd, 1977] D. Bobrow and T. Winograd. An overview of krl, a knowledge representation language. *Cognitive Science*, 1(1):3–46, 1977.

[Boyer and Moore, 1979] Robert S. Boyer and J Struther Moore. *A Computational Logic.* ACM Monograph Series. Academic Press, 1979.

[Brachman *et al.*, 1983] R. Brachman, R. Fikes, and H. Levesque. Krypton: A functional approach to knowledge representation. *IEEE Computer*, 16:63–73, 1983.

[Brachman, 1983] R. J. Brachman. What is-a is and isn't: An analysis of taxonomic links in semantic networks. *IEEE Computer*, 16(10):30–36, October 1983.

[Constable *et al.*, 1985] R.L. Constable, T. B. Knoblock, and J. L. Bates. Writing programs that constuct proofs. *Journal of Automated Reasoning*, pages 285–326, 1985.

[Downey *et al.*, 1980] Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations on the common subexpression problem. *JACM*, 27(4):758–771, October 1980.

[Fahlman, 1979] Scott E. Fahlman. *NETL: A System for Representing Real World Knowledge.* MIT Press, 1979.

[Ketonen, 1984] Jussi Ketonen. Ekl - a mathematically oriented proof checker. In *Proceedings of the Seventh International Conference on Automated Deduction*, pages 65–79, 1984.

[Kozen, 1977] Dexter C. Kozen. *Complexity of Finitely Presented Algebras.* PhD thesis, Cornell University, 1977.

[McAllester, 1989] David A. McAllester. *Ontic: A Knowledge Representation System for Mathematics.* MIT Press, 1989.

[Nelson and Oppen, 1980] Greg Nelson and Derek Oppen. Fast decision procedures based on congruence closure. *JACM*, 27(2):356–364, April 1980.